Federated Learning & Data Privacy, 2024-2025

Second Lab - 4 February 2025

Welcome to the second lab session of the Federated Learning & Data Privacy course! In our first lab, we implemented the Federated Averaging (FedAvg) algorithm, writing the client and aggregator classes, and we performed some preliminary experiments.

RECAP OF EXERCISE 3 - The Effect of Local Epochs

Objective: Analyze how the number of local epochs affects the model's performance in a federated learning setting.

Experiment:

- We ran FedAvg for different numbers of local epochs (e.g., 1, 5, 10, 50, 100).
- We recorded the test accuracy for each setting.

Plot:

- We plotted the local epochs on the x-axis and test accuracy on the y-axis.

Analysis:

- Discuss how local epochs influence model accuracy.
- Were you expecting this result?
- How was the data generated and partitioned in TP1? Justify your answer by examining data/mnist/generate_data.py and data/mnist/utils.py.

NEW EXERCISES FOR TP2

Goal: In this lab, we will analyze the effects of data heterogeneity, implement client sampling, and explore personalization within federated learning frameworks. A bonus exercise offers the opportunity to deploy a federated learning algorithm in a real, distributed network environment.

To get started, clone the TP2 folder from the lab repository.

EXERCISE 4.1 - The Impact of Data Heterogeneity

Objective: Demonstrate that an increase in the number of local epochs can potentially degrade FedAvg's performance under non-IID data distributions.

- **Preliminary question**: What non-IID data distribution means? Provide examples.
- **Pathological Split**: Familiarize yourself with the concept of "pathological split" as explained in [Communication-Efficient Learning of Deep Networks from Decentralized Data (Section 3) (https://arxiv.org/abs/1602.05629)](https://arxiv.org/abs/1602.05629). The pathological_non_iid_split function has been implemented for you in data/mnist/utils.py. Review this method and summarize it briefly.
- **Experiments**: Run the generate_data.py script with the --non_iid flag and set --n_classes_per_client=2 to partition the MNIST dataset in a non-IID fashion.
- **Plot**: Run experiments to observe how varying the number of local epochs (e.g., 1, 5, 10, 50, 100) influences the model's test accuracy under non-IID data distribution. Plot the relationship between

the number of local epochs and test accuracy.
- **Interpretation**: Briefly comment the results. Were these results expected?

EXERCISE 4.2 - Tackling Data Heterogeneity with FedProx

Objective: Understand how the FedProx algorithm addresses the challenges posed by data heterogeneity in federated learning and compare its performance with the FedAvg algorithm.

- **FedProx Overview**: FedProx modifies the local training objective by introducing a proximal term, which aims to reduce local model drift by penalizing significant deviations from the global model. Review the FedProx algorithm Federated Optimization in Heterogeneous Networks (Algorithm 2) (https://arxiv.org/abs/1812.06127) and our implementation of the ProxSGD class in utils/optim.py.
- **Experiments**: To initiate FedProx experiments, run the train.py script with local_optimizer = "prox_sgd" and set the proximal term coefficient mu = 2.
- **Plot**: Replicate the plot from Exercise 4.1, this time evaluating FedProx algorithm.
- **Analysis**: Discuss the observed differences in performance between FedAvg and FedProx. Are there specific configurations (e.g., number of local epochs) where FedProx particularly outperforms FedAvg?

EXERCISE 5 - Client Sampling

Objective: Implement two client sampling strategies from the research paper "On the Convergence of FedAvg on Non-IID Data" (https://arxiv.org/abs/1907.02189).

EXERCISE 5.1 - Uniform Sampling Without Replacement

Background

Understand uniform sampling as described in Assumption 6. This involves selecting a subset of clients $|S_t| = K$ at each round without replacement. Understand the aggregation formula given by $w_t \leftarrow \frac{N}{K} \sum_{k \in S_t} p_k w^k_t$.

Instructions

1. In aggregator.py, complete the sample_clients() method to uniformly sample self.n_clients_per_round clients from the total available clients.
2. Use self.rng.sample to sample self.n_clients_per_round unique ids from a population ranging from 0 to self.n_clients - 1.
3. Assign the list of sampled ids to self.sampled_clients_ids.
4. Modify the mix() method to:
   - Use only the sampled clients for training. For local training, loop over self.sampled_clients_ids instead of all clients.
   - Aggregate updates from the sampled clients. Adjust weights accordingly.

Run the code

Run the train.py script with sampling_rate = 0.2.

EXERCISE 5.2 - Sampling With Replacement

Background

Understand sampling with replacement according to sampling probabilities $p_1, \dots, p_N$. The aggregation formula adjusts to $w_t \leftarrow \frac{1}{K} \sum_{k \in S_t} w^k_t$.

Instructions

1. Extend the sample_clients() method to support sampling with replacement based on self.sample_with_replacement flag.
2. If self.sample_with_replacement is True, use self.rng.choices to sample clients based on their weights self.clients_weights.

Run the code

Run the train.py script with sampling_rate = 0.2 and sample_with_replacement = True.

Good luck, and don't hesitate to ask questions and collaborate with your peers!

At the end of the lesson, you can send your document and code to: francesco.diana@inria.fr (mailto:francesco.diana@inria.fr)